EECS 1012: LAB 08 – Code Breaker

A. IMPORTANT REMINDERS

1) You must attend your assigned lab session (we will be marking your submission in the lab).
2) You are required to arrive on time: anyone later than 15 minutes may not be admitted to the lab.
3) You are required to complete the pre-lab mini quiz posted on Moodle in the first 15 minutes of your lab time.
4) Each lab including the pre-lab mini quiz is about 1.5% of your overall grade.
5) TAs are in the lab to help you. They will also verify and mark your work at the end. Signal a TA for help if you stuck on any of the steps below. Yet, note that TAs would need to help other students too.

B. IMPORTANT PRE-LAB WORKS YOU NEED TO DO BEFORE GOING TO THE LAB

1) Complete your Learning Kit Project (that you started from Lab03 and added 20 problems to it recently) with 20 **more** problem definitions, flowcharts, and JavaScript solutions. Don't need to do any task for the "another solution" panel yet. As you will need this project for Lab09 and Lab10 too, failing to complete it by Lab08 may affect your grades in Lab09 and Lab10 too.

2) Download this lab files and read them carefully to the end.
3) If you are not familiar with the Code Breaker board game, visit https://en.wikipedia.org/wiki/Mastermind_(board_game). Note that the one we make in this lab has 5 code pegs (not 4).
4) You should have a good understanding of
    - Document Object Model https://www.w3schools.com/js/js_htmldom.asp , in particular `createElement`, append, etc. https://www.w3schools.com/jsref/dom_obj_document.asp
    - jQuery https://www.w3schools.com/jquery/
    - Revisit Slide 09-16 to 09-18 and Sides 09-23 to 09-29 of the lecture notes and identify the jQuery commands and review their meaning. We highly encourage you—prior to go to labs this week—to do two sets of exercises on jQuery selectors and events available in w3schools, starting here: https://www.w3schools.com/jquery/exercise_jq.asp?filename=exercise_jq_selectors1
    - JSON https://www.w3schools.com/js/js_json_intro.asp , in particular `stringify`, and `parse` methods. https://www.w3schools.com/js/js_json_stringify.asp and https://www.w3schools.com/js/js_json_parse.asp
5) Understanding AJAX `request` and `response` is also an asset, even though we are not going to use it directly. This topic will be clearer by end of next lab. https://www.w3schools.com/js/js_ajax_http_send.asp and https://www.w3schools.com/js/js_ajax_http_response.asp .

C. GOALS/OUTCOMES FOR LAB

1) To practice more computational thinking algorithms
2) To become familiar with JQuery and JSON
3) To work more with DOM

D. TASKS

1) TASK 1: Complete your Learning Kit with 40 flowcharts and JS codes. You may also want to enhance the implementation of your Learning Kit project by using `JQuery` and DOM. This task has 50% of your grade.
2) TASK 2: Client-Side of the Code Breaker Games. This task has the other 50% of your grade.

E. SUBMISSIONS

1) Manual verification by a TA

You will need to have one of the TAs of your lab to verify your work before submission. The TA will look at your various files in their progression. The TA may ask you to make minor modifications to the lab to demonstrate

your knowledge of the materials. The TA will mark your name off a list; You are required to sign the list to show your attendance and that you have been verified.

2) Moodle submission

Create a **folder** named "**Lab08**" and copy **all** of your HTML and JS files; Once you are done, compress the folder and upload the zip (or tar) file to moodle. Also, create a folder name "**LearningKit40**" and copy all your Learning Kit materials; then, compress them and upload the zip/tar file to moodle.

F. FURTHER DETAILS

Task 1) In the Learning Kit that you started in Lab03 and enhanced it in Lab07, you have provided the flowcharts and JavaScript codes for 20 problems. In this lab, you should increase the number of problems to 40 with the following requirements:

A) at least 7 problems should have a *loop* without nesting.
B) at least 3 problems should have a *nested loop*.
C) at least 2 problems should have a nested loop with depth of minimum two.
D) at least 3 problems should be related to *arrays*.
E) at least 5 problems should call *functions*.

Examples of algorithms requiring one loop without nesting:

1) Sum of numbers 10 to 30
2) Factorial
3) Fibonacci
4) a*b, without using multiplication (assume *a* and/or *b* are whole numbers)
5) a modulo b, without using modulo operation (use subtraction instead)
6) converting a number from base 10 to base 2.

Examples of algorithms requiring loops with one level of nesting:

1) draw triangle (Exercise 15 in Lab5)
2) draw upside down triangle
3) draw a diamond
4) output the multiplication table of size *n*
5) output a^b, without using power or multiplication
6) factorial, without using multiplication

Examples of algorithms requiring loops with two levels of nesting:

1) multiply two matrices
2) sum of numbers in a 3D array

Examples of algorithms using arrays:

1) examples on Slides 6-11 to 6-16
2) Simple Lottery: define an array of 3 items, store a random number between 0 to 9 to each item; ask a user to guess 3 numbers between 0 to 9 (the order is important); if all 3 guesses are good, bingo!

Examples of algorithms using functions:

1) An example on Slide 6-21
2) Output factorials of 1 to 10
3) Sum of all prime numbers less than 100
4) Output all numbers between 1 and 1000 that all palindrome

Recommendation: Instead of creating 40 buttons statically/explicitly in your HTML file, you may want to create the buttons in your `JavaScript` file by using `JQuery` and `DOM`.

In particular, in `myLearningKit.html`, delete all buttons from the nav tag:

```
    <nav>
      <!-- In Ex2, add an attribute onclick to button-->
      <button>Problem01</button>
      <!-- In Ex4, add an attribute onclick to button-->
      <button>Problem02</button>
      <button>Problem03</button>
      <button>Problem04</button>
      <button>Problem05</button>
      <button>Problem06</button>
      <button>Problem07</button>
      <button>Problem08</button>
      <button>Problem09</button>
      <button>Problem10</button>
    </nav>
  </header>
```

delete these

And add the following code fragment to your JS file:

```
for (var i = 1; i<=40; i++){

    var newBtn = document.createElement("button");
    $(newBtn).attr("id", "btn" + i);
    $(newBtn).html("problem " + i);

    $("nav").append(newBtn);
}
```

Note: we assign an id to each button.

We leave it for you to figure out how and where exactly this code can be added. Also, you need to figure out how to add handlers for "on click" events to these buttons.

To enhance your code a bit further, if you like to have more meaningful captions for each button, you may want to create an array like the example below.

```
var caption=["Odd or Even", " Sum of Numbers", "Triangle Area",
             "Sum of 10 to 30", "Decimal to Binary", "Quadratic Equation", ... ];

for (var i = 1; i<=40; i++){

    var newBtn = document.createElement("button");
    $(newBtn).attr("id", "btn" + i);
    $(newBtn).html(caption[i-1]);

    $("nav").append(newBtn);
}
```

Show your Learning Kit to your TA.

Note. You should not have any problem to find 40+ problems from what has been covered in lectures, labs, and labtests, midterm, and CTC sessions. However, if you need more problems, you may want to refer to the Course Kit.

Task 2: Use html, CSS, and JS to design the client side of the code-breaker game. We have provided you with a starter code, but you can design with any style you wish. If you like to use our starter code, you should:

1) Open `code_breakerV0.html` and save it as `code_breaker.html`. You should read the comments in your `code_breaker.html`, and make changes such that your html file becomes similar to `code_breakerV1.html`, eventually. Note that your html file instead of having 150 lines is going to have 28.

2) Also, open `code_breaker_clientV0.js` and save it as `code_breaker_client.js`. In the `createGameBoard` function of `code_breaker.js`, you should read the comments and write the code for all 15 lines specified by "`//...`". These lines dynamically build the html tags that you just deleted from your html file: you are creating those tags via your js file at run time (dynamically).

If you make the above changes properly, your code_breaker should work fine, that means you can start playing the game by opening your `code_breaker.html` in Firefox. We make sure that our `code_breaker_server` is up on `indigo.eecs.yorku.ca` until March 19, 2020. In the next lab, you will complete your own `server_side` code such that you be able to run it on your computer any time.

Show your code-breaker code to your TA.

Note that the CodeBreaker project is a great source of learning. Hence, read all comments carefully and make sure you have a clear understanding of it.

Next week, we will complete this project by adding the server component to it.

G. AFTER-LAB WORKS (THIS PART WILL NOT BE GRADED)

In order to review what you have learned in this lab as well as expanding your skills further, we recommend the following questions and extra practices:

1) It's good to start thinking to add an equivalent solution to your Learning Kit project. We highly recommend you pick `Java` as your next programming language. In the 3rd panel of your learning kit provide a screen shot of your Java code. Some students may prefer to use another language, such as C++, C, C#, etc.
2) You may want to learn a bit more about AJAX `request` and `response`. We will use them indirectly in the next lab. https://www.w3schools.com/js/js_ajax_http_send.asp and https://www.w3schools.com/js/js_ajax_http_response.asp
3) You may also want to start learning about `express JS`, as we are going to use it for the server side of CodeBreaker in next lab.

Please feel free to discuss any of these questions in the course forum or see the TAs and/or Instructors for help.